Improving static ordering of BDDs for reachability analysis

J. Vidal, D. Deharbe, D. Borrione

Improving static ordering of BDDs for reachability analysis

Jorgiano Vidal David Déharbe
DIMAp/UFRN, Natal, RN, Brazil

Dominique Borrione
TIMA, Grenoble, France

Abstract

Binary decision diagrams (BDDs) are used for automatic synthesis and formal verification of combinational and sequential circuits. However, a larger adoption of these technologies for sequential designs still depends on a more efficient use of BDDs. One important factor is the order of the variables in the BDD, which has a direct impact on the space (and time) requirements of the reachability algorithms. As the problem for finding the best order is NP-complete and requires, in practice, static and dynamic ordering heuristics, we introduce a new approach for finding an initial order for BDDs involved in the design of sequential circuits. This approach, called the *weighting heuristics*, has been implemented in the VIS toolset and favorably compares with state-of-the art heuristics.

1 Introduction

Binary Decision Diagrams [3] - BDDs - are an efficient data structure used for representing and manipulating boolean expressions : they use less space than the traditional methods, and provide mostly linear algorithms to operate over the expressions.

BDDs have been used for the verification and synthesis of combinational and sequential circuits [5, 4]. One important aspect of these techniques is the *reachability analysis*, that is, computing the characteristic function of all the states of the system that are reachable from a given (set of) initial configurations. It involves a breadth-first search that consists in iteratively calculating the successor set of a set of states, until a fixpoint is reached. The computation of the successor set is called the image computation and is one of the most expensive operations involved in these techniques.

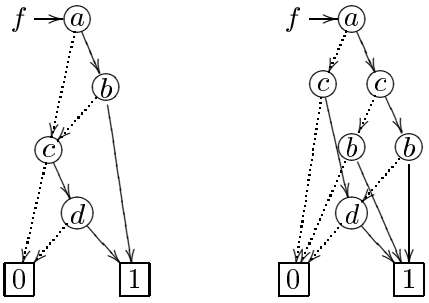

Figure 1: The same function with different orders

It is well-known that different orders on the BDD variables for a given function usually result in different BDD sizes. For example, consider the BDD for the function $f = (a \wedge b) \vee (c \wedge d)$. If the variable order is $a < b < c < d$ then the graph has 4 nodes; if b and c are swapped, then the graph has 6 nodes (figure 1). Consequently, swapping two adjacent variables results in a graph 50% larger than the original one.

Different algorithms have been developed for solving the BDD variable ordering [12, 8, 9, 6, 1], but none is focused on the reachability problem. In this paper, we introduce a new approach dedicated to reachability analysis, called the *weighting heuristics*, which uses the structure of the state machine. The main idea of the heuristics is to assign a weight to the variables and order the variables in decreasing order of weight.

This paper is organized as follows. Section 2 reviews the ordering problem and the algorithms that have been developed. Section 3 introduces

the weighting heuristics: first, we examine BDD ordering for reachability analysis, then we show how to order using the weighting heuristics. Section 4 shows the results of applying the weighting heuristics on VIS, a tool for verification and synthesis of sequential circuits. The paper concludes in section 5 with a discussion of our results.

2 State of the art

Traditionally, the BDDs variable ordering problem has been subdivided into two subproblems:

1. static ordering consists in finding an initial variable ordering

2. dynamic ordering – or reordering – means adapting the variable order during the manipulation of the BDDs.

It is worth mentioning that dynamic variable ordering heuristics are much more efficient and useful if the order before the reordering is reasonably good; so the idea of the reordering heuristics is to improve the existing order. More precisely, we observed [7], in reachability analysis applications, that the better the order before reordering, the better the order after reordering. In some cases, initiating with too poor an initial order (e.g. random) made it impossible to complete the reachability analysis within our space and time constraints. We conclude that, for an optimized use of BDDs, we need a strategy that includes both good static ordering and dynamic reordering heuristics.

In the remainder of this section, we present the most widely used static ordering heuristics, most of them were initially developped for BDDs representing combinational circuits [12, 8] and were later adapted to also handle sequential circuits.

2.1 Appending

Malik [12] introduced an algorithm for BDDs initial ordering that represents combinational circuits with only one output. This heuristics, called appending, is based on a post-order depth-first traversal of the circuit, where the root of the search is the circuit output, and the combinational elements (logic gates and inputs) are the tree nodes.

2.2 Interleaving

The previous appending algorithm was the basis for the development of the *interleaving based algorithm* [8]. This method, developed for combinational circuits with more than one output, is based on applying the appending algorithm successively to each output, constructing an overall order. When an output is processed, the new found nodes are inserted at the beginning of the order or, after their predecessor in this order, when already present.

2.3 Sampling scheme

The sampling scheme [9] consists in function analysis over the structure being represented, not the analysis of the structure itself. This method chooses some functions, called the samples, builds the BDDs for the samples, reorders the samples using some reordering algorithm (usually sifting [11]) and builds the BDDs that represents the whole structure based on the order found for the samples. Thus, expensive but precise methods are applied only in reduced portions of the system.

2.4 Parallel genetic algorithm

Costa et al. [6] developed a parallel genetic algorithm for finding an initial variable ordering for combinational circuits. Each individual is a candidate order, and the quality of an individual is the size of the BDDs for this order, where less nodes means better order.

Different populations of individuals evolve independently and in parallel, and at some point of the process, there is an interchange between populations. The algorithm stops if there are n iterations without improvement of the population.

This algorithm has found very good orders, but it is very expensive and its principal application would be logic synthesis of combinational circuits.

2.5 BDD ordering for interacting FSM

This algorithm, introduced in [1], handles sequential circuits. It considers a set of interacting finite state machines and orders the machines based on their dependencies. Variables in a machine are

grouped together and may have any order between them.

The order is decided by analysing the communications graph of the structure for minimizing the communication complexity. Very good results were obtained for the construction of the transition relation and reachability analysis.

3 Weighting heuristics

Static ordering finds a BDD variable ordering before the effective use of BDDs. After construction of the graph, reachability analysis requires the continuous construction and destruction of the representation of new functions. As a consequence, the graph size may changes dramatically, causing an explosion on the number of nodes. Finding a good BDD variable ordering that minimizes the changes of the size is the big challenge for reachability analysis. We identified two steps for reachability analysis: build the transition relation and find the reachable states. After these two steps, a reordering algorithm should be called to minimize the graph size.

The transition relation of a circuit characterizes all the transitions. It makes it possible to compute the set of states reachable in one transition from a given set of states. The next value of a flip-flop $x_i \in X$ is given by function $f_i(X, I)$, the *transition function* of x_i, over the inputs I and flip-flops X of the circuit. The transition relation of the flip-flop is: $R(X, I, X') = x'_i \Leftrightarrow f_i(X, I)$, where $x'_i \in X'$ is the next value of x_i.

The *transition relation* of the circuit is obtained by the conjunction of the transitions of all flip-flops:

$$R(X, I, X') = \bigwedge_{i=0}^{n} x'_i \Leftrightarrow f_i(X, I).$$

Reachability analysis computes all states reachable in any number of steps from the initial states. It is defined as the following *least fixpoint*, where S_0 characterizes the initial states:

$$\mathbf{lfp}K[S_0(X') \vee \exists X, I(K(X) \wedge R(X, I, X'))]$$

The weighting algorithm aims at finding an order that allows the completion of reachability analysis within reasonable space bounds. To do this, the heuristics uses the state machine structure to order

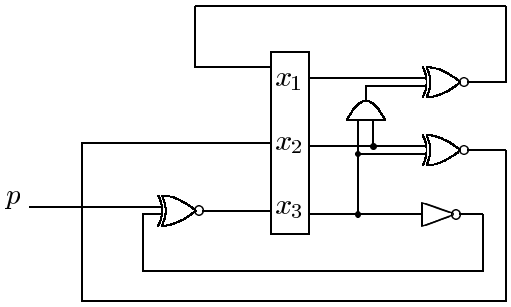

Figure 2: modulo 8 counter

the variables, assigning weight to the variables and ordering the variables in decreasing order of the weight. The weight of a variable x is the number of times x is a parameter of the transition function of a flip-flop. Therefore, if x is present in the arguments of k functions, then k i is the weight of x. Intuitively, if a variable is an argument for many functions, then we can tell that x will have a great influence on the machine behavior, and that it should appear early in the variable ordering, being one of the first decision variables in the BDDs [7].

Algorithm 1 describes how to calculate the variable order using the weighting heuristics.

Algorithm 1 Weighting heuristics

```
begin
  foreach w_i ∈ X, I do
    weight(w_i) ← 0
  od
  foreach x_i ∈ X do
    S_i ← support(f_i)
    foreach w_j ∈ S_i do
      weight(w_j) ← weight(w_j) + 1
    od
  od
  order(X ∪ I, weight)
  comment: In decreasing order
end
```

To demonstrate the algorithm, consider the circuit 2, the transition relation functions are the following:

$$f_1(W) \quad = \quad (x_3 \wedge x_2) \oplus x_1$$

$$f_2(W) = x_3 \oplus x_2$$
$$f_3(W) = \neg x_3 \oplus p$$

and the variables weights are:

x_1	1
x_2	2
x_3	3
p	1

so, from the weights obtained, we have the following order:

$$\{x_3, x_3', x_2, x_2', x_1, x_1', p\}$$

4 Experimental results

The weighting heuristics has been integrated in a tool for verification and synthesis developed at Berkeley University, called VIS [2].

VIS represents a circuit as a graph, where each element (combinational or memory) is a node and the node are interconnected. We adapted the weight computation, where now we look at a node n and find all the other nodes that have connections to n. The weight of all nodes begins with 0, for each node n that belongs to a set of nodes that has connections to the node being processed, the weight of n is incremented. The node to be processed for finding the dependencies are all the output and register nodes.

Two versions of the algorithm were implemented : the first one called weight1 (W1) stops seeking the node dependencies if it finds a register or an input of the circuit ; and the other one called weight2 (W2) stops seeking only at the inputs of the circuit. Therefore, W1 counts the weight in the dependency relation between variables, whereas W2 reports the weight in the transitive closure of this dependency relation.

The tests were realized in four phases, and for each phase we compared the time and the number of BDDs nodes after completion of the phase. The four phases are: static order (SO), build the transition relation (TR), reachability analysis (RA) and dynamic reordering (R). The tests were done over a set of descriptions that are deployed with the VIS [2] distribution. We used 17 descriptions for testing.

The elapsed computation time of each phase of the test was the same for all algorithms and the difference was on the number of BDDs nodes for each phase. Table 1 shows a summary of the results. For each phase, we report the number of best results for each initial ordering heuristics.

	W1	W2	A	I	ML	MR
TR	0	2	4	1	8	2
RA	2	4	4	1	2	4
RO	0	6	3	3	3	2

Table 1: Number of better results after each phase.

The weight algorithms W1 and W2 were compared against the append (A), interleaving (I), merge left (ML) and merge right (MR).

We can observe that after building the transition relation, the merge left algorithm is the best, but the results change during the other phases. When reachability is done, three algorithms have the same results, including one of the weighting heuristics, this shows that the algorithms are quite equivalent. But if we reorder the BDD after reachability, which is advocated in the most recent results on ordering strategy for reachability analysis [10], we note that the weighting algorithm yields the best results. We believe that this happens because the focus of the approach is the finite state transition machine analysis.

The main idea of BDD variable ordering for reachability analysis is to keep the size of BDDs in reasonable sizes, combining initial ordering heuristics with the invocation of dynamic reordering between the different phases of the process. Figure 3 shows the size of the BDDs, normalized by the Weight1 algorithm to show the peak number of nodes until the reachability analysis. We can observe that Weight2 and append algorithms are better than the others. Only in one example the Weight (Weight1 and Weight2) methods are worse than all the others and also in one example, the weighting algorithms are better than all the others.

In these experiments, the computation times were independent of the chosen heuristics, showing no significant difference. However, in an environment with more severe memory constraints, runs exceeding the available main memory size would have resulted also in a blow-up in execution time: BDD-based applications cannot take advantage of secondary memory, due to the lack of locality they

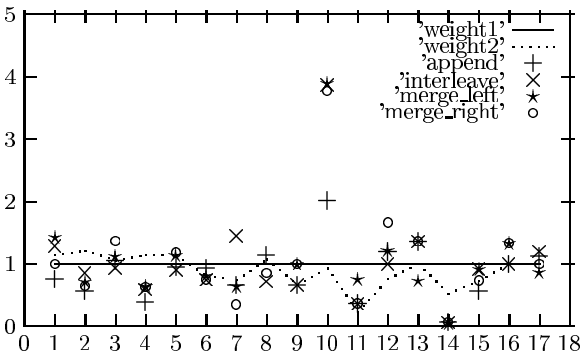

Figure 3: Peak number of BDD nodes until the reordering after reachability analysis

show in their patterns of memory access.

5 Conclusions and future work

This paper proposes a new approach to the static ordering of sequential circuits, focused on reducing the bottleneck of space requirements needed to complete reachability analysis, which is an important component of both formal verification (symbolic model checking and equivalence checking), and logic synthesis. As we observed, the main results of our initial ordering heuristics in reachability analysis is the enhancements it causes to the reordering during the whole process, obtaining in the end the best results among a series of heuristics implemented into a state of the art academic tool.

Good experimental results have already been obtained, but we believe the algorithm may still be improved. One issue that can be addressed is that of variables with the same weight - how should they be ordered? We also plan to investigate experimentally the impact of the heuristics in the actual verification and synthesis of sequential circuits.

References

[1] A. Aziz, S. Tasiran, and R.K. Brayton. BDD Variable Ordering for Interacting Finite State Machines. In *31st DAC*, 1994.

[2] R. Brayton, G. Hachtel, A. Sangiovanni-Vincentelli, and F. Somenzi. Vis: A system for verification and synthesis. In *CAV'96*, volume 1102 of *LNCS*, 1996.

[3] R.E. Bryant. Graph-based algorithm for boolean function manipulation. *IEEE Transactions Computers*, C(35):1035–1044, 1986.

[4] J.R. Burch, E.M. Clarke, K.L. McMillan, and D.L. Dill. Sequential circuit verification using symbolic model checking. In *27th DAC*, 1990.

[5] E.M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons for branching time temporal logic. In *Logics of Programs: Workshop*, volume 131 of *LNCS*, pages 52–71, 1981.

[6] U. Costa, D. Déharbe, and A. Moreira. Advances in BDD reduction using parallel genetic algorithm. In *Proceedings of the* 10^{th} *International Workshop on Logic Synthesis (IWLS'01)*.

[7] D. Déharbe and J. Vidal. Optimizing BDD-based tools analysing variable dependencies. In *SBCCI'01*. Computer Society Press, 2001.

[8] H. Fujii, G. Ootomo, and C. Hori. Interleaving Based Variable Ordering Methods for Ordered Binary Decision Diagrams. In *ICCAD'93*, pages 38–41, Santa Clara, California.

[9] J. Jain, W. Adams, and M. Fujita. Sampling schemes for computing OBDD variable orderings. In *ICCAD'98*, pages 631–638.

[10] G. Kahmi and L. Fix. Adaptive variable reordering for symbolic model checking, 1998.

[11] R. Rudell. Dynamic Variable Ordering for Ordered Binary Decision Diagrams. In *ICCAD'93*. IEEE Computer Society Press, 1993.

[12] S. Malik, A.R. Wang, R.K. Brayton, and A. Sangiovanni-Vincentelli. Logic Verification Using Binary Decision Diagrams in a Logic Synthesis Environment. In *ICCAD'88*.